Journal of Computer Virology and Hacking Techniques
https://doi.org/10.1007/s11416-018-0319-9

ORIGINAL PAPER

Security implications of running windows software on a Linux system using Wine: a malware analysis study

Rory Duncan[1] · **Z. Cliffe Schreuders**[1]

Received: 13 April 2017 / Accepted: 22 March 2018

Abstract
Linux is considered to be less prone to malware compared to other operating systems, and as a result Linux users rarely run anti-malware. However, many popular software applications released on other platforms cannot run natively on Linux. Wine is a popular compatibility layer for running Windows programs on Linux. The level of security risk that Wine poses to Linux users is largely undocumented. This project was conducted to assess the security implications of using Wine, and to determine if any specific types of malware or malware behavior have a significant effect on the malware being successful in Wine. Dynamic analysis (both automated and manual) was applied to 30 malware samples both in a Windows environment and Linux environment running Wine. Behavior analyzed included file system, registry, and network access, and the spawning of processes, and services. The behavior was compared to determine malware success in Wine. The study results provide evidence that Wine can pose serious security implications when used to run Windows software in a Linux environment. Five samples of Windows malware were run successfully through Wine on a Linux system. No significant relationships were discovered between the success of the malware and its high-level behavior or malware type. However, certain API calls could not be recreated in a Linux environment, and led to failure of malware to execute via Wine. This suggests that particular malware samples that utilize these API calls will never run completely successfully in a Linux environment. As a consequence, the success of some samples can be determined from observing the API calls when run within a Windows environment.

Keywords Malware analysis · Malware compatibility · Linux · Wine

1 Introduction

The relative security of operating systems has long been the subject of debate. Linux compares favorably to Windows and Mac OS X in terms of security features, such as mandatory access controls and file permissions. It has also been suggested that the availability of source code allows Linux bugs and security related programming issues to be easily spotted and patched by the community [18]. Linus's Law states that "given enough eyeballs, all bugs are shallow" [11]. Although malware exists that targets Linux systems, this is relatively rare: the majority of malware targets Windows systems (in part due to targeting of market share). Furthermore, Linux software repositories provide a trusted source for software installation, which also reduces the likelihood of malware on Linux. Common practice is to not run anti-malware software on Linux systems.

However, many software developers do not create their software for Linux, and choose to focus on creating packages for operating systems with the highest market shares, such as Windows, which has traditionally held the highest market share of desktop PCs and laptops. This results in Linux users needing compatibility layer software if they wish to run Windows software on their Linux operating system. Wine is the defacto compatibility layer for running Windows applications on POSIX-compliant operating systems, such as Linux, Mac OS X, and BSD.

Wine works by translating functions that are Windows specific into a format that can be understood by Linux (or other POSIX systems) [19]. The use of compatibility layer software may present a security threat to Linux users. The fact that it enables Windows software to run in Linux environment introduces the possibility of running Windows malware on an otherwise secure Linux machine. The level of security risk that Wine poses to Linux users is largely undocumented.

✉ Z. Cliffe Schreuders
c.schreuders@leedsbeckett.ac.uk

[1] Cybercrime and Security Innovation (CSI) Centre, Leeds Beckett University, Headingley Campus, Leeds LS6 3QS, UK

Published online: 26 April 2018

This paper describes a study conducted to investigate the level of threat that the use of Wine potentially presents to Linux systems, and to determine if any specific types of malware or malware behavior have a significant effect on the malware being successful in Wine. In this study, thirty samples of Windows malware were run in Windows and then run on a Linux system using Wine. Dynamic analysis (both automated and manual) was compared to assess how successful each sample of malware was in terms of running in the Linux environment.

2 Literature review

2.1 Malware analysis

The term malware refers to malicious software created with the intention of compromising a computer system or destroying data [16]. Davis et al. (2009) note that malware potentially offers malicious coders a large return on the investment of time spent developing malware and that this is possibly a reason why malware is such a widespread threat in the computing field [4]. Elisan (2012) also believes that the reason that malware is so popular among cyber-criminals is because of the potentially high financial gain [5]. Elisan (2012) proceeds to comment that malware analysis plays a vital role in computer security as in order to defend computer systems against threats from malware it is essential that knowledge is gained into exactly what the malware is doing on a system [5].

The term dynamic analysis refers to the process of running a malicious sample of code on a computer system with the intention of studying its behavior by recording API calls, processes and network activity [7]. Marak (2015) comments that the term static malware analysis was coined to cover analysis techniques that involve viewing the binary code of a malicious sample, which is often done using disassembler software [7].

Malin et al. (2012) state that using a mixture of static analysis and dynamic analysis is the most effective form of malware analysis [8]. This view is also confirmed by Seo et al. (2014) who comment that both dynamic and static analysis have limitations, to overcome these limitations it is essential to use a combination of both techniques [13].

2.2 Malware analysis techniques

Elisan (2012) states that static analysis would seem like the obvious choice for malware in terms of ease and efficiency; however, he also notes that the results gathered during a static analysis are less useful as the malware is inactive when analyzed [5]. The observations of Ahmadi et al. (2013) on static analysis, hold a similar view to Elisan on using static analysis and comment malicious coders can utilize a diverse range of techniques that make static analysis of malware inaccurate, they mention that the techniques used include entry point obfuscation code packing and control flow [1]. Vidyarthi (2015) confirms this view expressing that if a sample of malware is packed, encrypted, complex or a large sample a static analysis can become very difficult [2]. Shijo et al. (2015) also agree with this, stating that polymorphic and metamorphic samples fail in a static analysis as do any malware samples that have been created using obfuscation techniques with the intention of making them harder to analyze [15]. They state that dynamic analysis of malware is not affected by these techniques; this makes it vital in malware analysis. Sharif et al. (n.d) agree with this view and mention that because of techniques that effect static analysis of code such as obfuscations and packers, dynamic analysis is generally the technique adopted by software designed to automate malware analysis [14].

Shijo et al. (2015) note that the main drawbacks to dynamic analysis are that each sample must be run individually in a sandboxed testing environment, which is time-consuming, and that the results of a dynamic analysis may be inaccurate as malware may behave different in a secure environment and some malware may wait for certain dates or times before executing, which means some of the functions of the malware may not be detected [15]. This is agreed by Liangboonprakong et al. (2013) who also comment that the infection of the test system could be dependent on vulnerable software being installed in the test environment [6]. The project described in this paper focused mainly on the behavioural dynamic analysis of malware; as the aim of the project was to determine if malware behaves differently in a particular environment, static analysis is less relevant.

2.3 Malware analysis tools

Ligh et al. (2010) recommends the software Anubis for analyzing Windows binaries; further research concluded this was not a suitable choice for the project as the software has been discontinued [7]. Shijo et al. (2015) note that a useful tool for dynamic malware analysis is Cuckoo Sandbox, this program generates a report that contains a list of API calls made by the malware when it was executed in a secure environment [15]. Provataki et al. (2013) remark that the software Cuckoo Sandbox is a new piece of software that is written in Python, they state that the software is fast and efficient when performing malware analysis. It has built-in characteristics that ensure stealth when the malware is running in the secure environment, this ensures it remains undetected by the malware running in the analysis environment and the results should be very accurate [10].

Vasilescu et al. (2014) remark that a range of websites can be used as dynamic analysis tools, these are however

generally an inaccurate method of dynamically analyzing malware as they have limited registry keys and installed applications [17]. Examples of online dynamic analysers are Anubis and Malwr, an analyser based on Cuckoo Sandbox. Online analysis environments were not used in the project as it is imperative that accurate results are gathered from the dynamic analysis. The accuracy of the results of online analysis programs is questionable as the malware may attempt to utilize software or write to registry keys that are not present on the system, causing the API calls to be unsuccessful. Cuckoo Sandbox was used in the project as it is free open source software and it has a large list of features that aided in the dynamic analysis of malware. Cuckoo Sandbox also has a useful feature that scans a file against multiple anti-virus programs, this can be used to determine if the malware is detected by ClamAV (one of the most popular Linux anti-malware suites).

The most relevant tool to analyze the effects of malware in Linux was Zero Wine. Vasilescu et al. (2014) note that Zero Wine is a valuable tool for analyzing Windows applications in Wine. The software would be a useful tool for this project as it allows a dynamic analysis of Windows malware running in Wine [17]. Vasilescu et al. (2014) also comment that Zero Wine will only run applications that are designed to run on Windows, this is useful as during the analysis stage of the project it was paramount to choose samples that will run in Zero Wine [17]. Malin et al. (2012) mention that Zero Wine Tryouts is a branch of the original Zero Wine project and that this version has more features integrated into the software [8]. For this research, Zero Wine Tryouts was the malware analysis framework of choice for the Linux environment testing.

2.4 Malware on wine and related research

Codeweavers (2008) comment that the use of Windows emulation or compatibility layer software can make an otherwise secure system such as a Linux-based operating system vulnerable to Windows malware [3]. Moen (2005) created a brief report on some of his findings when running Windows malware in a Linux environment using Wine [9]. The findings in Moen's report support Codeweavers (2008) comments as one of the malware samples appeared to have successfully run in Wine. Overall the majority of the malware did not appear to have run successfully which would imply there is a low success rate with malware that can actually compromise a Linux computer when running in Wine. There is a substantial gap in the research literature regarding the relevance of these warnings, and whether malware samples will actually run in Wine and be capable of compromising a Linux system. This lack of research was a key motivation for this project to provide further insight.

3 Aims

The aim of this malware analysis study was to assess the security implications of running Windows software on a Linux system using Wine. The research focused on analyzing the effects of Windows malware within a Linux environment running Windows compatibility software. The study aimed to analyze how different malicious samples behaved in a Windows environment and a Linux environment through Wine. The collected results from the two environments were compared, this comparison was used to determine the success of each malware sample when running in Wine, overall and in terms of categories of behaviour(filesystem, registry, processes, networking, services). The research project collected this data to assess infection success factors, including if any relationships could be discovered between specific types of malware or malware behavior and the malware being successful when running in Wine. Relationships were assessed using statistical tests on the data collected at the analysis stage of the project. The project also aimed to develop recommendations for Linux users who choose to use the software Wine, the results of the study were also used to determine the threat level that Wine poses to Linux users.

The working hypothesis was that high-level behavioral characteristics and file attributes of malware samples would have a large effect on the malware's ability to run successfully through Wine. The behaviors being assessed were: the malware making changes to the file system, making changes to the registry, starting processes, network traffic, the number of files dropped/created, and the number of processes spawned. The file attributes that were being assessed were; the file size of the malware, the compile date of the malware and the category of the malware.

4 Methods

4.1 Malware samples

The 30 malware samples collected for use during this study were downloaded from online malware repositories. The primary sources that were used were the malware sharing websites Contagio, KernelMode.info, Malware.lu and VX Vault. These were selected as the primary sources as they had an extensive selection of malicious software that is available for download, this approach also ensured variety in the types of malware analyzed. Ten malware samples were collected from Contagio and Kernelmode.info, four from Malware.lu. Six samples were also acquired from VX Vault to ensure that more recent (and before analysis, unidentified) samples were included in the study.

4.2 Analysis environment

Two malware analysis environments were required. The first environment was necessary to test the malware in a Windows environment; this would be used to establish how the malware acts when it is run on a vulnerable computer system. The second malware analysis environment was used to run the malware in Linux. Analysis results were collected from both environments; theses results contained details about the samples behavior in different areas including registry, processes, network, file system and services. The testing results from both environments were compared to assess if the malware had behaved the same in Wine as it did in the Windows environment.

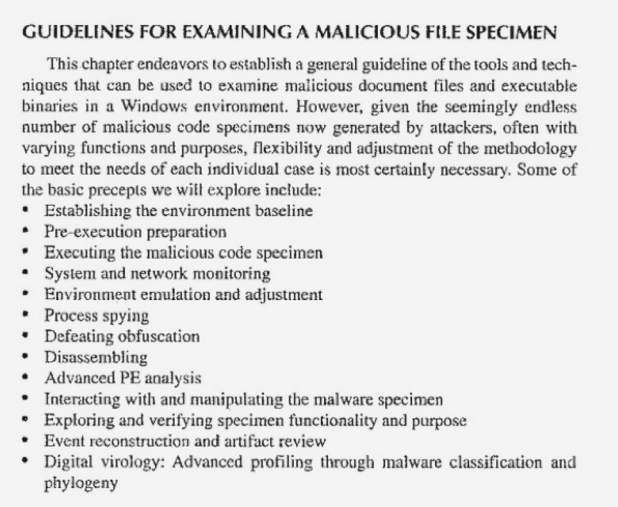
The software chosen to analyze the malware in a Windows environment was Cuckoo Sandbox. The software is written in Python and has the ability to analyze malware in an automated manner [10]. Cuckoo Sandbox allows malware to run in a sandboxed environment and then provides a detailed report that contains information on the malware's activity in multiple areas including network, services, file system, registry and processes.

Zero Wine Tryouts was also used in this study, this software was utilized as its primary function is to analyze how Windows malware behaves in a Linux environment running Wine [17]. The software records the API calls that are executed through Wine, Zero Wine Tryouts then creates a report detailing this information which can be used to assess the behavior of the malware and to identify the calls made on the system [20]. Other tools (such as debuggers and system call tracers) were used to further inspect processes and verify results from the automated analysis tools.

4.3 Analysis methodology

A methodology was developed to establish continuity between the analysis of each sample and guarantee that all necessary data was collected. The image documented in Fig. 1 demonstrates an extract from the book Malware Forensics Field Guide for Windows Systems, the extract details steps to analyzing malware. This was one source that was utilized in order to create a malware analysis methodology. Some of the steps noted in this source were impractical to implement into the project, for example doing an advanced static examination of the portable executable file is not appropriate for this project as the focus of this work is studying the behavior of malware in a Linux environment. A static analysis may still be appropriate to depict details such as the compile date of the malware.

Further research was completed into a framework for the malware analysis testing, a useful methodology was discovered on the Secfence website. The methodology is



GUIDELINES FOR EXAMINING A MALICIOUS FILE SPECIMEN

This chapter endeavors to establish a general guideline of the tools and techniques that can be used to examine malicious document files and executable binaries in a Windows environment. However, given the seemingly endless number of malicious code specimens now generated by attackers, often with varying functions and purposes, flexibility and adjustment of the methodology to meet the needs of each individual case is most certainly necessary. Some of the basic precepts we will explore include:

- Establishing the environment baseline
- Pre-execution preparation
- Executing the malicious code specimen
- System and network monitoring
- Environment emulation and adjustment
- Process spying
- Defeating obfuscation
- Disassembling
- Advanced PE analysis
- Interacting with and manipulating the malware specimen
- Exploring and verifying specimen functionality and purpose
- Event reconstruction and artifact review
- Digital virology: Advanced profiling through malware classification and phylogeny

Fig. 1 Guidelines for examining a malicious file specimen [8]

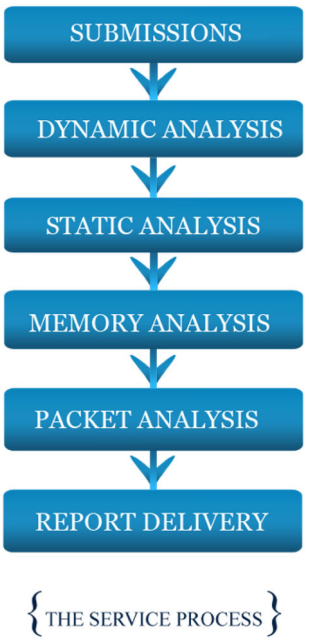


Fig. 2 Secfence malware analysis methodology [12]

demonstrated in Fig. 2. Although the steps are vague it gives a detailed overview of the steps that should be taken in a successful analysis. Some of the stages in this methodology again are not suitable for the project, for example, no memory analysis was performed during this project. The reason that memory analysis was not be focused on during the analysis stage of this project is that the malware analysis environment Zero Wine that was used to analyze malware within a Linux environment has limited features to capture and analyze the memory of a process.

Static Analysis of the sample Cuckoo Sandbox features integrated static malware analysis functions; these functions were used in order to perform a static analysis of the malware samples that were being analyzed. Statically analyzing each sample individually was an essential part of the study, the information that was collected during the static analysis was later used to assess relationships between the success of a malware sample running in Wine and it's file size, file type and compile time. The main static analysis tool used in the project was Cuckoo Sandbox, this software automatically does a static analysis and outputs the results in the final report, along with the behavioral analysis results. As this was the software of choice for the behavioral analysis it was a logical choice to also use it for the static analysis. Static analysis was used to extract file details including:

- File name
  This was extracted from the sample via static analysis to determine what the original file name of the malware sample was.
- File size
  The file size was extracted using the static analysis features of Cuckoo Sandbox.
- File type
  The file type was also extracted using Cuckoo Sandbox's static analysis features.
- MD5 Hash
  The MD5 hash was collected to determine the uniqueness of each sample in the study. It was important that different samples were used throughout and no sample was used twice. The MD5 hash ensured that no two files that were identical were used more than once in the analysis stage of the project.
- Compile Time
  The compile time of the sample was collected using Cuckoo Sandbox's static analysis features. The compile time was not always accurate as some dates were very clearly wrong, indicating that the metadata of the file had been tampered with. The compile time was collected to determine if the age of the malware had any effect on its ability to run successfully in Wine.

Behavioral analysis of the sample The behavioral analysis involved five categories of analysis for each sample

- File system: The file-based API calls, along with any files that are dropped by the sample.
- Registry: Registry based API calls.
- Processes: The processes that were spawned and documenting any crashes.
- Network: The network traffic.
- Services: Any services started by the malware.

The comparison of each sample was done by running a Python script to process the results from the Windows environment, this script extracted all of the API calls of interest from the Cuckoo Sandbox report. This included API calls related to file system changes, registry changes, process starting, services starting and network related API calls. The signatures section of Cuckoo Sandbox would also be observed as this is an implemented function of Cuckoo Sandbox that extracts key malicious API calls during the analysis. The API calls were then compared to the API calls that were recorded by Zero Wine Tryouts in the Linux environment. For each significant API call a search was conducted for similar API calls in the Zero Wine report, which was used to determine if the malware had similar behavior in Wine when compared to the Windows environment and hence determining how successful the malware was at running in Wine. If a call was made multiple times in Windows the number of times it appeared in the Cuckoo Sandbox was recorded, this could then be compared to the Zero Wine report to ensure that the call appeared a comparable number of times in the Linux environment. The PCAP files that were generated by Zero Wine Tryouts and Cuckoo Sandbox were also compared by hand to determine if similar packets were observed in both environments.

5 Results

5.1 Analysis results

This section reports on the results from each of the malware samples that were analyzed.

5.1.1 Narilam sample analysis

Narliam is an information stealing Trojan that is usually distributed in spam emails and malicious websites; after infecting a system it attempts to gather information about a system, such as the operating system, running processes, IP address and a list of running processes; file details can be viewed in Table 1. The results of the analysis, which are summarized in Table 2, provided strong evidence to suggest that the malware had been successful at compromising the Linux computer running Wine. The file and registry based API calls were very similar in both environments. The malware was copied to C:\WINDOWS\system32\lssas.exe, compromising the lssase.exe service in both Linux and Windows. The malware was also able to install itself via Wine, making registry based system changes to ensure it would automatically run when the computer was started. The malware started all the processes that were present in the Windows process tree. The findings suggest that the malware has been successful as the behavior in Windows and Linux were alike.

Table 1 Narilam sample details

Malware family	Narilam
Malware type	Worm
File name	data.exe
Size (bytes)	1639284
File type	PE32 (GUI) Intel 80386
Compile date	03/09/2009

Table 2 Narilam sample success details

Files success	True
Registry success	True
Process success	True
Network success	N/A
Services success	N/A
Overall success	True

Table 3 Hikit sample details

Malware family	Hikit
Malware type	Rootkit
File name	oci.dll
Size (bytes)	262656
File type	PE32 (DLL) (console) Intel 80386
Compile date	20/06/2011

5.1.2 Hikit sample analysis

Hikit is designed to create a backdoor into the infected machine, this backdoor enables an attacker to steal information from the victim computer. The attacker can also send commands and additional malware to the infected machine using the backdoor created by the Hikit rootkit. Further details of the Hikit sample can be viewed in Table 3. None of the files that appeared to be dropped by the malware in the Windows environment were also dropped in the Linux environment, this suggests that the malware had not been successful at compromising the Linux system. None of the relevant registry API calls that were made in Windows were successful in Wine, this indicates that the malware had failed to make key registry changes when running in Linux. In the Linux environment, the main process failed to initialize which is likely the reason no file system or registry based API calls of interest were found in the Wine activity. The DNS requests made by the malware when it was running in a Windows environment did not happen when the malware was run in Linux using Wine. Neither of the two services opened in Windows appeared to be opened in the Linux environment. Overall it was clear that the malware had very little, if any, success at running in the Linux environment and had not compromised the Linux system, Table 4 confirms this as the malware was unsuccessful at recreating the behavior observed in the Windows environment in all areas.

Table 4 Hikit sample success details

Files success	False
Registry success	False
Process success	False
Network success	False
Services success	False
Overall success	False

Table 5 Stabuniq sample details

Malware family	Stabuniq
Malware type	Trojan
File name	Stabuniq_F31B797831B36A48(...)
Size (bytes)	79360
File type	PE32 (GUI) Intel 80386
Compile date	21/03/2012

Table 6 Stabuniq sample success details

Files success	N/A
Registry success	N/A
Process success	False
Network success	False
Services success	False
Overall success	False

5.1.3 Stabuniq sample analysis

Stabuniq is a Trojan designed to steal information. The malware is normally distributed in spam emails and malicious websites. Details of the sample can be viewed in Table 5. After infecting a system, it attempts to gather information about a system, such as the operating system, running processes, IP address and a list of running processes and then sends this information back to a remote server. The evidence collected during analysis suggests that the malware failed to compromise the Linux system running Wine. In Linux, the second process failed to initialize and the main process crashed during execution. The sample was packed using Armadillo, this is a probable cause of the samples failure to run as it was found that Zero Wine fails to execute Windows PE programs packed in Armadillo. Table 6 documents that the malware failed to recreate the behavior detected in the Windows environment in all applicable areas.

5.1.4 Drixed sample analysis

The Drixed malware family samples are known to steal banking credentials, leak sensitive data, and provide remote access

Table 7 Drixed sample details

Malware family	Drixed
Malware type	Trojan
File name	43s5d6f7g.exe
Size (bytes)	196608
File type	PE32 (console) Intel 80386
Compile date	14/12/2015

Table 8 Drixed sample success details

Files success	False
Registry success	False
Process success	False
Network success	False
Services success	False
Overall success	False

Table 9 Batch wiper sample details

Malware family	Batch Wiper
Malware type	Trojan
File name	GroveMonitor.exe_
Size (bytes)	185928
File type	PE32 (GUI) Intel 80386
Compile date	04/01/2011

Table 10 Batch wiper sample success details

Files success	False
Registry success	True
Process success	False
Network success	N/A
Services success	N/A
Overall success	Partially

to attackers, and are typically spread via malicious Microsoft Office macros attached to spam email. Static analysis determined that original filename of the malware exe file was Winchat.exe. Winchat.exe is a Windows system program, which is typically found in C:\Windows\System32. As in this case, when found in another location it is possible that the file is malicious. Analysis results for the Drixed sample can be viewed in Table 7. The analysis suggests that the malware was unsuccessful in compromising a computer running Linux and the Windows compatibility software Wine, the overall results can be viewed in Table 8. The malware did not extract information from internet browsers when running in the Linux environment. The file that was dropped in Windows was not dropped in the Linux environment. The malware did not fingerprint the machine when running in Linux and did not configure itself to auto-run on start-up. The malware did start the main processes but this crashed during analysis. In the Linux environment, the malware did not make any similar network related API calls to the one made in Windows. The network traffic to host 199.7.136.84' was not detected when the malware was run in Linux, suggesting the malware's call home function failed in this environment. The service opened in Windows was not opened in the Linux environment. Overall it is clear that the malware failed to compromise the Linux system.

5.1.5 Batch wiper sample analysis

Batch Wiper is a Trojan horse that is used by attackers to delete all files in a partition and it also removes the directory of the user profile. When the malware runs successfully on a system, it creates the files sleep.exe, juboot.exe and jucheck.exe. The file details of the sample can be observed in Table 9. Multple files that were dropped in the Windows environment were not dropped by the malware in the Linux environment. The malware called eight functions to delete files in the Windows but only called three in the Linux. Similar registry based API calls were made in both environments, the most significant call being found in both environments in the location Software\Microsoft\Windows\CurrentVersion-\Run, this registry location was changed to allow the malware to run when the infected computer started up. When running in Zero Wine some key processes failed to start and the Sleep.exe' process crashed. Overall it was clear that the malware was not successful at compromising the Linux computer system. Registry changes appeared the same in the Linux environment suggesting the malware was partially successful at running on the Linux system. It was successful at installing itself to run at start-up. An overview of the samples success at running in Linux can be viewed in Table 10.

5.1.6 Dialer sample analysis

Dialer malware is a type of spyware that after infecting a system attempts to dial telephone lines, this can often lead to very expensive telephone bills. When the malware was run in the Linux system key files were not dropped; sample details of the malware can be viewed in Table 11. When the malware was run on Windows it dropped the file rasphone.pbk was dropped by the malware, the difference list that was generated by Zero Wine confirmed that this file was not dropped in the Linux environment. The malware also failed to make significant changes to the registry in Linux such as the RAS AutoDial which is located at Software\Microsoft\RAS AutoDial. The malware successfully started all of the processes in the Linux environment; however, it was unable to start any of the services that were started in the Windows environment.

Table 11 Dialer sample details

Malware family	Dialer
Malware type	Spyware
File name	91f805aece0434de667ac79a0fd4(...)
Size (bytes)	14720
File type	PE32 (GUI) Intel 80386
Compile date	08/06/2005

Table 12 Dialer sample success details

Files success	False
Registry success	False
Process success	True
Network success	N/A
Services success	N/A
Overall success	False

Table 13 MyDoom sample details

Malware family	MyDoom
Malware type	Worm
File name	weaver.email.FileDownloadLocation
Size (bytes)	34612
File type	PE32 (GUI) Intel 80386
Compile date	01/01/1970

The malware clearly was not successful when run in Linux, Table 12 documents the areas the malware was successful in.

5.1.7 MyDoom sample analysis

MyDoom is a piece of malware that generally spreads over email or peer to peer networks. The worm is used to create a backdoor known as Zincite, which listens on TCP port 1034. After infecting a computer, the worm can spread further using mailing lists found on the computer that it has compromised. Details about the sample can be observed in Table 13. Only three of the eight files that were dropped in Windows were dropped on the Linux system and the lsass.exe file had a different hash when dropped the Linux environment, revealing that it was not an identical file to the one dropped in Windows. The malware did not recreate the vital registry based API calls that were called in the Windows environment. The malware was unsuccessful at making changes to the registry to ensure that it ran on start-up. The malware also failed to make changes to the registry to edit the Internet settings. The same processes started in Windows and Linux. The network traffic was not similar in the Linux environment with no DNS requests being detected. None of the services started in Windows were also started in Linux. It is clear that overall the malware was not successful at compromising the Linux system. An overview of the results for the MyDoom malware sample can be viewed in Table 14.

Table 14 MyDoom sample success details

Files success	False
Registry success	False
Process success	True
Network success	False
Services success	False
Overall success	False

Table 15 Minamps sample details

Malware family	Minamps
Malware type	Trojan
File name	Minaps_C99FA835350AA9(...)
Size (bytes)	647599
File type	PE32 (GUI) Intel 80386
Compile date	16/08/2009

Table 16 Minamps sample success details

Files success	True
Registry success	False
Process success	True
Network success	True
Services success	False
Overall success	Partially

5.1.8 Minamps sample analysis

Minamps is a backdoor remote access Trojan. The malware is generally dropped by other pieces of malware when they infect a machine; however, Minamps is also able to compromise machines if the user was to accidentally download it from a malicious website. Details about the sample used in this analysis can be viewed in Table 15. All of the files that were dropped in the Windows environment were also dropped in the Linux environment. There were some differences in the recorded behavior in that key registry locations have not been edited, which suggests that the malware was not completely successful in compromising the Linux system. Similar processes were initialized in both Linux and Windows. The network-based API calls were similar in both Windows and Linux. However, the service RASMAN did not start in the Linux environment. It is clear that this sample of malware was partially successful at running in the Linux environment. The file-based changes were very similar, all of the relevant processes have been initialized and similar network traffic was detected to the network traffic identified in the Windows environment test. Table 16 documents the overall results of this malware sample.

Table 17 PlugX, Korplug sample details

Malware family	PlugX, Korplug
Malware type	Remote Access Trojan
File name	Plugx_00fdb6ad7345c091(...)
Size (bytes)	241622
File type	PE32 (GUI) Intel 80386
Compile date	15/02/2010

Table 18 PlugX, Korplug sample success details

Files success	False
Registry success	False
Process success	True
Network success	False
Services success	N/A
Overall success	False

5.1.9 PlugX, Korplug sample analysis

PlugX or Korplug is a form of Trojan that enables remote access to an infected machine. The malware is typically used for the purpose of information theft. Table 17 displays information related to the PlugX, Korplug malware sample. The file system based API calls showed the relevant files were created on the system in both Linux and Windows. These files were not present in the Zero Wine difference list, which means they were not present on the Linux system, which suggests the malware had not been successful. A mutex was created in both Linux and Windows (with differing names). The registry-based API calls were different in Linux: the malware failed to install itself to run at start-up, and it did not collect information from the computer such as the computer name. Although no running processes appeared to have failed to remain running, the fact that the network traffic differed in Linux suggests that the malware has not worked as expected. Table 18 provides overall details of the malware success at running in Linux.

5.1.10 Wykcores sample analysis

Wyckores is a Trojan Dropper. It is known to use the software backdoor known as Backdoor Murcy. All of the file-based API calls were similar between Windows and Linux and all the files dropped in Windows were dropped in Linux. The sample details can be viewed in Table 19. The main registry locations were changed in both Windows and Linux suggesting that the malware was able to start Backdoor Mercy in Linux. The processes that were started were the same in both Windows and Linux environments. The network traffic was also comparable, further supporting that the malware was successful in the Linux environment. Wykcores dropped

Table 19 Wykcores sample details

Malware family	Wykcores
Malware type	Trojan Dropper
File name	Wykcores_0D38D6C2B9EB81(...)
Size (bytes)	73484
File type	PE32 (GUI) Intel 80386
Compile date	19/06/1992

Table 20 Wykcores sample success details

Files success	True
Registry success	True
Process success	True
Network success	True
Services success	True
Overall success	True

Table 21 Didrex sample details

Malware family	Didrex
Malware type	Trojan
File name	34frgegrg.exe
Size (bytes)	314880
File type	PE32 (GUI) Intel 80386
Compile date	03/02/2016

a malicious file on the system that appeared to provide backdoor access to the infected systems. The CyService service was also started in both environments. Overall the malware has been successful at running in the Linux environment, this is confirmed by the overall results in Table 20.

5.1.11 Didrex sample analysis

Dridex is a derivative of the Cridex banking malware family, which typically spreads via malicious Microsoft Office macros sent via spam email. Once compromised it joins a botnet and modifies the victims web browser to steal banking credentials. Details of this sample are displayed in Table 21. The recorded activity of the malware being run in Linux was very different to when it was run in Windows. None of the file based or registry based API calls of interest that occured in the Cuckoo Sandbox environment was in the Zero Wine activity. This means that it has not achieved its function of fingerprinting the system and had not made changes to the system to ensure that it would run on start-up. The fact that the main process appears to have failed to initialize suggests further that the malware has failed to compromise the target machine. The network activity, in Windows recorded as attempting to communicate with 91.239.232.145, was also not present in the PCAP file generated in the Linux environ-

Table 22 Didrex sample success details

Files success	False
Registry success	False
Process success	False
Network success	False
Services success	False
Overall success	False

Table 23 Dozmot sample details

Malware family	Dozmot
Malware type	Trojan Dropper
File name	Dozmot.D2190db2c50c6ceb(...)
Size (bytes)	30208
File type	PE32 (DLL) (console) Intel 80386
Compile date	04/05/2011

Table 24 Dozmot sample success details

Files success	False
Registry success	N/A
Process success	False
Network success	N/A
Services success	N/A
Overall success	False

ment. Neither the Tapisrv' nor Rasman' service was opened in the Linux environment. Table 22 concludes that the malware was unsuccessful in all areas, meaning it has failed to run successfully.

5.1.12 Dozmot sample analysis

Dozmot.D is a malicious program used for the purpose of stealing passwords for online games accounts, file details of the sample can be observed in Table 23. The malware did not call many API functions in the Windows environment, and when it was run in Linux even fewer API calls were present in the Zero Wine activity. In Windows the malware only called one successful API function that referred to the file system this was the NtOpenFile function that was called to the location C:\DOCUME-1\LOCALS-1\Temp\Dozmo.. . The main process crashed when this malware was run in the Linux environment with the error message 'Unable to find the entry point L"DllMain'. The malware was unsuccessful in all areas, failing to run, this is confirmed in Table 24.

5.1.13 Potao sample analysis

Potao is a remote access Trojan, which is designed and utilized for data theft. The form of malware is most common in Ukraine, Russia, Georgia and Belarus, it is generally used for targeted espionage. Attackers typically use social engineering, or exe wrappers, in order to execute a successful attack with the malware. The sample of Potao's file details can be viewed in Table 25. The recorded activity indicated that the malware was not successful at compromising the Linux system. The malware did not drop any files in the Linux environment, in contrast to the two dropped files in the Windows environment. In the Linux environment the file pikbw.b file was not loaded using the LdrLoadDll function. The LdrGetProcedureAddress was called on both systems with some similar calls, this function was called eighty times more on the Windows system. It was clear that the malware was unsuccessful at running in Linux, this is confirmed in the overall results, presented in Table 26.

Table 25 Potao sample details

Malware family	Potao
Malware type	Backdoor Trojan
File name	Potao_1stVersion_0C7183D761(...)
Size (bytes)	59904
File type	PE32 (GUI) Intel 80386
Compile date	01/10/2006

Table 26 Potao sample success details

Files success	False
Registry success	N/A
Process success	True
Network success	N/A
Services success	N/A
Overall success	False

Table 27 Gamarue sample details

Malware family	Gamarue
Malware type	Worm
File name	Gamarue.F.exe
Size (bytes)	32768
File type	PE32 (GUI) Intel 80386
Compile date	30/04/2005

5.1.14 Gamarue sample analysis

Gamarue is a worm that acts as a remote access Trojan and steals information. It propagates via removable drives and is also known to spread via other malware and spam emails. Table 27 documents file details of the sample. The file system based API calls that were found in the Cuckoo Sandbox report were not present in the Zero Wine activity logs. The only API call that referred to the file system of the Windows machine was the NtOpenFile function being called

Table 28 Gamarue sample success details

Files success	False
Registry success	True
Process success	False
Network success	False
Services success	N/A
Overall success	Partially

Table 29 TDL/Alureon sample details

Malware family	TDL/Alureon
Malware type	Rootkit
File name	DNSChanger_0d7b87223d6fd2(...)
Size (bytes)	186368
File type	PE32 (GUI) Intel 80386
Compile date	09/03/2005

to the registry location 'C: \WINDOWS\system32 \wuauclt.exe' . Registry based API calls were comparable with the RegOpenKey function being called to the subkey 'Drive' in the HKEY_CLASSES_ROOT hive in both Windows and Linux. An internal process that was created in Windows did not appear to be created in the Linux environment. The network traffic that was detected in Windows revealed multiple DNS and ICMP requests. No similar packets were detected when the malware was run in the Linux environment. Overall the malware sample was partially successful at running in the Linux environment as similarities were found in the registry based API calls. The overall results are presented in Table 28.

5.1.15 TDL/Alureon sample analysis

TDL/Alureon is a backdoor rootkit that monitors network traffic to steal information, such as credentials. This sample is known to change DNS settings on the system after infection. Sample details are displayed in Table 29. No files were dropped by the malware in either Windows or Linux. The Cuckoo Sandbox report outlined that the NtCreateFile was called to the file 000069b9.tmp, the malware was then copied into this file, similar calls were not found in the Zero Wine report. No registry based API calls were made in Windows or Linux. The main process that was spawned in the Linux environment crashed with an UnhandledExceptionFilter, and the malware was not successful at compromising the system. The network traffic that was detected when the malware was run in the Windows environment was substantially different to the network traffic that was discovered in the Linux environment. When the malware was run in the Linux environment, no NBNS packets were detected, further suggesting that the malware failed in the Linux environment. The overall results

Table 30 TDL/Alureon sample success details

Files success	False
Registry success	N/A
Process success	False
Network success	False
Services success	N/A
Overall success	False

Table 31 SC-KeyLog sample details

Malware family	SC-KeyLog
Malware type	Trojan Dropper
File name	SCKeyLog.O_bf53d17ace80(...)
Size (bytes)	29460
File type	PE32 (GUI) Intel 80386
Compile date	15/09/2004

Table 32 SC-KeyLog sample success details

Files success	True
Registry success	True
Process success	True
Network success	True
Services success	N/A
Overall success	True

presented in Table 30 show that the sample was unsuccessful in all areas, resulting in it failing to run successfully in the Linux environment.

5.1.16 SC-KeyLog sample analysis

SC-Keylog is a Trojan used by attackers to log the keystrokes on an infected machine; sensitive information such as passwords can be stolen using this method [4]. Details about the sample used in this analysis can be viewed in Table 31. The main file system based API calls of interest were recreated in the Linux environment. The files that were dropped on the Windows system were also dropped on the Zero Wine virtual machine: the hashes of these two dropped files were the same on both systems. In Linux an additional file was dropped by the malware, with the filename Cf Hack.dll'. Similar registry based API calls were found in both the Cuckoo Sandbox and Zero Wine environments. The relevant sub process was initialized in both Windows and Linux without failure. Although fewer DNS requests were found in the Linux environment when compared to the Windows environment, requests were sent to all the same domains. Table 32 displays the overall results of this sample. It is clear that the sample has been successful at running in all areas and was successful at running in the Linux environment.

Table 33 Wirenet sample details

Malware family	Wirenet
Malware type	Password Stealing Trojan
File name	Host.exe
Size (bytes)	61952
File type	PE32 (GUI) Intel 80386 (stripped)
Compile date	07/08/2012

Table 34 Wirenet sample success details

Files success	True
Registry success	N/A
Process success	True
Network success	True
Services success	N/A
Overall success	True

Table 35 Dyzap sample details

Malware family	Dyzap
Malware type	Spyware Trojan
File name	Document-772976_829712.scr
Size (bytes)	246784
File type	PE32 (GUI) Intel 80386
Compile date	22/05/1983

5.1.17 Wirenet sample analysis

Wirenet is a common form of malware that is used to steal passwords; variations of the malware also exist that target OSX and Linux machines. Table 33 documents some of the samples details. No findings of interest were discovered in the file system based API calls. A mutex was successfully created in both environments. All of the relevant processes were started in Linux. The network-based API calls that were present in Cuckoo were also executed in the Linux environment, these appeared to be failed attempts to connect to network sockets. No network traffic was detected in either environment. Table 34 reveals the overall results of the sample. These findings suggest that the malware was successful at running in Linux when compared to a Windows system.

5.1.18 Dyzap sample analysis

Dyzap is a form of spyware Trojan that is used for information theft, the malware is commonly used to steal banking details. The malware is most common in China with over half of known infections occurring there. Table 35 reveals some details about the sample used in this analysis. The malware did have some success at recreating some of the file system based API calls of interest; however, not all API calls of interest were executed in the Linux environment. The file userdata.dat' was dropped in the Linux environment with a different file name, the file Document-772976-829712.src' that was dropped in the Windows environment was not dropped in the Linux environment. None of the registry-based API calls were present in the Zero Wine logs. The fact that these registry-based API calls were not present in the Linux system report indicates that the malware was not successful in installing itself to run at start-up as in Linux no changes were made to the registry location Software\Microsoft\Windows\CurrentVersion\Run. In Linux the malware also failed to fingerprint the target as no RegQueryValue functions were found in registry locations such as the MachineGuid. All of the relevant processes were initialized in the Linux environment. Table 36 displays the overall results of the sample, it is clear that the sample has not been successful at running in the Linux environment.

Table 36 Dyzap sample success details

Files success	False
Registry success	False
Process success	True
Network success	N/A
Services success	N/A
Overall success	False

Table 37 CoreBot sample details

Malware family	CoreBot
Malware type	Banking Trojan
File name	781c6.exe
Size (bytes)	497152
File type	PE32 (GUI) Intel 80386
Compile date	08/12/2015

5.1.19 CoreBot sample analysis

The malware sample is a Trojan and is used by attackers in order to steal private information. Table 37 displays the details of the sample used in this study. None of the file-based API calls of interest took place within Wine. The expected mutexes were not created in the Linux environment. None of the files that were dropped in the Windows environment were droped in the Zero Wine environment. This further suggests that the malware was unsuccessful. The malware did not appear to fingerprint the Linux machine as it did when running in the Windows environment. This suggests that the registry based API calls were not successfully recreated in Linux as the value of the MachineGuid was not queried by the malware. None of the appropriate internal processes were started by the malware in the Linux environment. The net-

Table 38 CoreBot sample success details

Files success	False
Registry success	False
Process success	False
Network success	False
Services success	N/A
Overall success	False

Table 39 Kawpfuni sample details

Malware family	Kawpfuni
Malware type	Backdoor Trojan
File name	1f0469a08681198ff9288b(...)
Size (bytes)	585728
File type	PE32 (GUI) Intel 80386
Compile date	13/06/2009

work traffic that was detected in the Linux environment also differed greatly to the Windows network traffic, in Windows multiple DNS and NBNS requests were detected. These were not detected when the malware was running in the Linux environment. The overall results shown in Table 38 show that the sample has been unsuccessful in all areas and unsuccessful overall.

5.1.20 Kawpfuni sample analysis

Kawpfuni is a backdoor Trojan that is known to be used for military espionage. Details about the sample are documented in Table 39. The file system based API calls of significance were similar in both Linux and Windows. Although the dropped files did not appear in the Zero Wine difference list these were deleted from the system and is likely the reason they were not present in the difference list. The registry-based API calls of interest did not appear to be executed in the Linux environment. When the malware was run in the Window environment it appeared to make many changes in the Internet Settings key. The RegCreateKeyExA function was called in the locations Software\Microsoft\Internet Settings\5.0\Cache\History and Software\Microsoft\Internet Settings\5.0\Cache\Cookies. These are signs of the malware stealing sensitive information from the registry. No similar calls were found in the Zero Wine activity. All of the expected processes were started successfully in the Linux environment. It is clear that the malware has been partially successful at running in the Linux environment as the file based changes were the same in Windows and Linux, the overall results shown in Table 40 support that the sample has been partially successful.

Table 40 Kawpfuni sample success details

Files success	True
Registry success	False
Process success	True
Network success	N/A
Services success	N/A
Overall success	Partially

Table 41 Skypii sample details

Malware family	Skypii
Malware type	Worm
File name	6B8E96CC7ADAF886C7(...)
Size (bytes)	103424
File type	PE32 (GUI) Intel 80386
Compile date	06/06/2004

Table 42 Skypii sample success details

Files success	False
Registry success	N/A
Process success	False
Network success	N/A
Services success	N/A
Overall success	False

5.1.21 Skypii sample analysis

Skypii is a worm, the sample was downloaded from kernelmode.info. Table 41 displays details about the chosen sample. This malware spreads using the instant messaging service on the programs Windows Live Messenger and Skype [5]. A mutex called 'DBWinMutex' was created in the Windows environment but was not created in the Linux environment. No changes were made to the registry in either Linux or Windows. When the malware was run in Windows the malware used the CreateProcessInternalW function, this API call was used to start the process 6B8E96CC7ADAF886C7...exe. No similar process was started in Linux, this suggests that the malware failed in the Linux environment as it failed to initialise key processes. The malware then proceeded to inject code into this process. The malware used the WriteProcessMemory to achieve this, no similar API calls were found within the Zero Wine report. This further supports the argument that the malware was unable to compromise the Linux system running Wine. Neither Cuckoo Sandbox or Zero Wine detected any network traffic. Overall these results suggest that the malware failed to compromise the Linux system. The overall results are displayed in Table 42.

Table 43 4DW4R3 sample details

Malware family	4DW4R3
Malware type	Rootkit
File name	Rootkit_4rw3r3_load.ex_
Size (bytes)	52224
File type	PE32 (GUI) Intel 80386
Compile date	27/01/2010

Table 44 4DW4R3 sample success details

Files success	True
Registry success	N/A
Process success	True
Network success	False
Services success	N/A
Overall success	Partially

Table 45 Loki Bot sample details

Malware family	Loki Bot
Malware type	Spyware
File name	4213294.root_1_0.scr.ViR
Size (bytes)	156672
File type	PE32 (GUI) Intel 80386
Compile date	07/06/2014

Table 46 LokiBot sample success details

Files success	False
Registry success	False
Process success	False
Network success	N/A
Services success	N/A
Overall success	False

5.1.22 4DW4R3 sample analysis

Details about the selected sample can be observed in Table 43. On analysis, the malware installed a print processor, that acts as a backdoor to the system for attackers. All of the significant file-based API calls that were called in Windows were also called in the Linux environment. The difference list created by Zero Wine did not contain the file '17.tmp' that was dropped by the malware in the Windows environment, this was caused by the fact that the malware deleted this file during the infection. All relevant processes were initialized in the Linux environment. However, the malware was not entirely successful as it did not generate any network traffic in the Linux environment. The network traffic from the Windows virtual machine revealed that the malware attempted a DNS request to triplexfund.com. No similar packets were captured in the PCAP file generated by Zero Wine, this suggests that the malware was not completely successful as the call home function failed. Table 44 highlights that the sample did have a moderately high degree of success when running in the Linux environment and it was partially successful.

5.1.23 Loki Bot sample analysis

Upon infection, Loki Bot gathers and extracts passwords and crypto-currency wallets from a wide range of software. Details of the Loki Bot sample selected can be viewed in Table 45. The file system based API calls of interest that were found in the Cuckoo Sandbox reports were not present in the Zero Wine output. The files '5511af46f5325f67...' and 'autoexec.bat' that were created in the Windows environment also were not created in the Linux environment. The key registry related calls that were executed in the Windows environment were not executed in the Linux environment and there was no evidence of the malware installing itself to auto-run at start-up of the system in the Linux environment. The Cuckoo Sandbox report detailed tha the malware made changes to registry location Software\Microsoft\Windows NT\CurentVersion\Winlogon to ensure it would run when Windows started. No similar calls were found in the Zero Wine report. Key processes crashed in the Linux environment suggesting that the malicious sample had failed to run successfully. The malware used the CreateProcessInternalW function in Windows to create the process svchost.exe. When the malware was run in Wine it was apparent that the malware attempted to initialise this process, however the ExitProcess fuction was called before the process could make any changes. When running in Windows this process called one hundred and fifty two functions that attempted to unhook Windows functions. Table 46 highlights that the malware was unsuccessful in all areas when running in Linux, resulting in the malware being unsuccessful overall.

5.1.24 Nitol sample analysis

Nitol is a Trojan used by attackers to form a botnet of zombie computer systems, typically used for denial of service attacks. The details of the sample of Nitol used in this study including the file name, file type and file size can be viewed in Table 47 The only file-based API call that was present in the Cuckoo Sandbox report was also present in the Zero Wine, this was the malware being copied to the file path location C:\Windows\system32\'. The registry-based API calls that made changes to the Nationalkyd' service were also present in both Linux and Windows. All of the relevant processes were started in each environment. The network traffic was similar in both environments; however, more TCP requests were made in Windows and DNS requests were only found

Table 47 Nitol sample details

Malware family	Nitol
Malware type	Trojan
File name	Yy999.exe
Size (bytes)	24576
File type	PE32 (GUI) Intel 80386
Compile date	08/09/2013

Table 48 Nitol sample success details

Files success	True
Registry success	True
Process success	True
Network success	True
Services success	True
Overall success	True

Table 49 Nivdort sample details

Malware family	Nivdort
Malware type	Trojan
File name	sample.exe
Size (bytes)	892416
File type	PE32 (GUI) Intel 80386
Compile date	10/07/2013

in the Linux environment. The Nationalkyd' service was started in both Windows and Linux. The information shown in Table 48 concludes that overall the malware has been successful at compromising the Linux system.

5.1.25 Nivdort sample analysis

Nivdort is a particular malware family known to install to the Windows folder, which is done to evade detection from anti-malware software. Information extracted about the sample used in this study can be viewed in Table 49. Via Wine the malware was unable to extract private information from Internet browsers as it did on the Windows system. Key file-based API calls that were present in the Cuckoo Sandbox did not have similar calls in the Zero Wine environment. Key files were not created on the Linux system, such as the tst file. None of the registry based changes of interest were executed by the malware in the Linux environment. The machine was not fingerprinted, the malware did not install itself to run on start-up, and no changes were made to the Internet settings in Linux. The malware did not spawn all of the relevant processes in the Linux environment and a key process crashed. DNS requests to multiple domains were detected when the malware was run in Windows, no network activity was detected in Linux. The Shell Diagnostic Brightness

Table 50 Nivdort sample success details

Files success	False
Registry success	False
Process success	False
Network success	False
Services success	False
Overall success	False

Table 51 Unknown1 sample details

Malware family	Unknown1
Malware type	Trojan Dropper
File name	newversion.exe
Size (bytes)	5766144
File type	PE32 (GUI) Intel 80386 (stripped)
Compile date	01/11/1971

Receiver was started in the Windows environment but no record of the malware attempting to initialize this service could be found in the Linux activity. The malware also started the Rasman service in Windows, but this service was not started in Linux. As illustrated in Table 50, this sample of malware apparently failed in the Linux environment.

5.1.26 Unknown sample 1 analysis

As this sample was downloaded from VX vault, it's family is unknown. Table 51 highlights the details of the malware sample. Changes that were made to the file system to make the sample run at start-up were not recreated in the Linux environment. Some DLL files that were called in Windows were not called in the Linux environment. The NtCreateFile function was called in Windows at the location C:\Documents and Settings\Rory\Local Settings\Temp \Crd.vbe. This file was dropped by the malware. The CreateFileW function was also called in Zero Wine, to create the file Crd.vbe. This finding initially suggests some success as the malware has dropped a key file in the Linux environment. After further investigation into the two reports it could be determined that the NtCreateFile function was called to the file rsaenh.dll in Windows. No similar API calls were found in the Linux environment. The DLL file was then utilised by the malware using the LdrLoadDll in the Windows environment. No similar calls were found loading this DLL file in the Linux environment. The malware did not install itself for auto run at start-up in the Linux environment by making changes at the location C: \Documents and Settings \All Users \Start Menu \Programs \Startup. All of the processes required were spawned. Also, the malware did not make the same registry calls. Although it did extract the computer name from the registry using the GetComputerNameW function, in Win-

Table 52 Unknown1 sample success details

Files success	False
Registry success	False
Process success	True
Network success	N/A
Services success	N/A
Overall success	False

Table 53 Unknown2 sample details

Malware family	Unknown1
Malware type	Unknown
File name	647129ff9f491c81dc8298b6cb6a(...)
Size (bytes)	301125
File type	PE32 (GUI) Intel 80386
Compile date	29/01/2012

dows the 'NtQueryValueKey' function was used in the object attribute 'ActiveComputerName'. Overall it is clear the malware was not completely successful at running within the Linux environment. Table 52 confirms that the malware has not been successful.

5.1.27 Unknown sample 2 analysis

Information about this particular sample can be viewed in Table 53. The analysis determined that the files PSAPI.DLL, uxtheme.dll and SETUPAPI.dll all failed to be loaded in the Linux environment. The malware dropped a file called 0_Wrapper.log when run in the Windows environment. It was determined that this file was not dropped in the Linux environment as it was not present in the Zero Wine difference list. The malware did manage to start all of the processes in Linux that were started in the Windows environment. None of the changes to the registries made in Windows were also made when the malware was run in Zero Wine. The malware made changes to the Windows registry in the location Software\Microsoft\Windows\CurrentVersion\-Explorer\ MountPoints2\788d3620-9923-11e5-85dd-806d6172696f. No similar entries were found in the Zero Wine report, suggesting that the malware was not successful at compromising the Linux system. In the Windows environment the RegQueryValueExW function was used to extract data from the computer registry. No similar API calls were found in the Zero Wine report suggesting further that the malware failed in the Linux environment. As can be observered in Table 54 the sample was not successful when running in the Linux environment.

Table 54 Unknown2 sample success details

Files success	False
Registry success	False
Process success	False
Network success	N/A
Services success	N/A
Overall success	False

Table 55 Unknown3 sample details

Malware family	Unknown3
Malware type	Unknown
File name	firefox.exe
Size (bytes)	1138688
File type	PE32 (GUI) Intel 80386
Compile date	29/01/2015

Table 56 Unknown3 sample success details

Files success	False
Registry success	False
Process success	False
Network success	False
Services success	N/A
Overall success	False

5.1.28 Unknown sample 3 analysis

Details of the sample of malware are presented in Table 55. The malware was unable to create the mutex 'DC_ -MUTEX79VYXBE' in Linux, this mutex was created in the Windows environment. When the malware was run in the Windows environment very few registry related API calls were present in the Cuckoo Sandbox Report. The malware mainly appeared to be querying registry values. When the malware was run in the Linux environment no similar calls appeared to be present in the Zero Wine activity log. Only four processes were started in the Linux environment: three less than the seven that were started in Windows. When the malware was run in Linux no significant network traffic was detected, in Windows multiple ICMP requests and DNS requests were detected. The malware attempted DNS requests to two addresses when it was run in a Windows environment. These addresses were adipluto.dynu.com and plutorack.linkpc.net. When the malware was run in the Linux environment no similar DNS requests were found in the PCAP file. Overall the malware is not successful in the Linux environment. Table 56 documents an overview of the malware failing in all areas that were assessed.

Table 57 Unknown4 sample details

Malware family	Unknown4
Malware type	Unknown
File name	ozze.exe
Size (bytes)	893942
File type	PE32 (GUI) Intel 80386
Compile date	29/10/2012

Table 58 Unknown4 sample success details

Files success	True
Registry success	False
Process success	False
Network success	False
Services success	N/A
Overall success	Partially

5.1.29 Unknown sample 4 analysis

Table 57 documents what is known about this sample. The file-based API calls were comparable in Windows and Linux. A malicious executable file was created in Linux and Windows, however the naming format differed slightly. In Windows the name was 'ozze.exe', in Linux it was named 'ttWIc8.exe'. This malicious executable was then copied into the location \Application Data\lpm26v\lmp26v.exe on both the Linux system and Windows systems. The files that were dropped in the Windows environment were also dropped on the Linux system. When the malware was run in the Linux environment all three of the files that were dropped in Windows were also dropped by the malware. The hashes for pid.txt and Windows.lnk appeared to be different in Linux when compared to the hashes of the files dropped in Windows.

The significant registry based calls that were executed in Windows did not appear in the Wine activity. In Windows the malware called the RegCreateKeyExW in the registry location Software\Microsoft\Windows \CurrentVersion\Explorer\ MountPoints2, possibly in attempt to infect any external storage devices with malware in an attempt to spread the infection. No similar API calls were found in the Zero Wine report. When the sample was run in the Windows environment DNS requests to the domain vs.redirectme.net were detected. When the same sample was run in Zero Wine these DNS requests were not detected, thus suggesting that the malware has failed to compromise the Linux computer system. Overall it is clear that the sample has been partially successful as the file-based API calls and created files were comparable. The results for this analysis are summarized in Table 58.

Table 59 Unknown5 sample details

Malware family	Unknown5
Malware type	Unknown
File name	host9.exe
Size (bytes)	76800
File type	PE32 (GUI) Intel 80386
Compile date	19/06/1992

5.1.30 Unknown sample 5 analysis

This sample's malware family is unknown. Table 59 shows what is known about the sample. Two files were dropped when the malware was run in Windows ruta.txt and .Identifier, the NtCreateFile function was called in Windows to create the file ruta.txt and .Identifier. Neither of these files were dropped on the Linux system. In Windows the malware also created a mutex, utilising the NTCreateMutant function to create a mutex called RdwdIEit, when the malware was run in Zero Wine no mutex was created. The malware did not fingerprint the system in the Linux environment. In the Cuckoo Sandbox report API calls were found that suggested the malware collects information from the system in attempt to fingerprint it. The malware utilized the function RegQueryValueExA to find the GUID of the machine. This call was not replicated in the Linux environment. The malware also did not make registry related changes to ensure it runs on start-up of the system in the Linux environment. Seven processes were started in Windows and only two were spawned in the Linux environment. When the malware was run in the Linux environment it only started one other process explorer.exe, this process was not started in Windows. When the malware was run in Windows it attempted to connect to the domain windows00.duckdns.org. This is potentially the domain it would send all the stolen information to after the malware has collected it. When the malware was run in Zero Wine it was apparent that the malware had not attempted to connect to any domains. In the Linux environment none of the services that were started in Windows appeared. When the malware was run in the Windows environment it appeared to open the RASMAN service. No evidence of this service starting in the Linux environment was present in the Zero Wine report. As Table 60 concludes, the sample has been unsuccessful in all areas and hence has failed to run in the Linux environment.

5.2 Overall results

The results gathered from the malware analysis stage of the study were used to determine the overall success rates of Windows malware samples running in Wine. As illustrated in Fig. 3, of the thirty malware samples analyzed, 16.7%

Table 60 Unknown5 sample success details

Files success	False
Registry success	False
Process success	False
Network success	False
Services success	False
Overall success	False

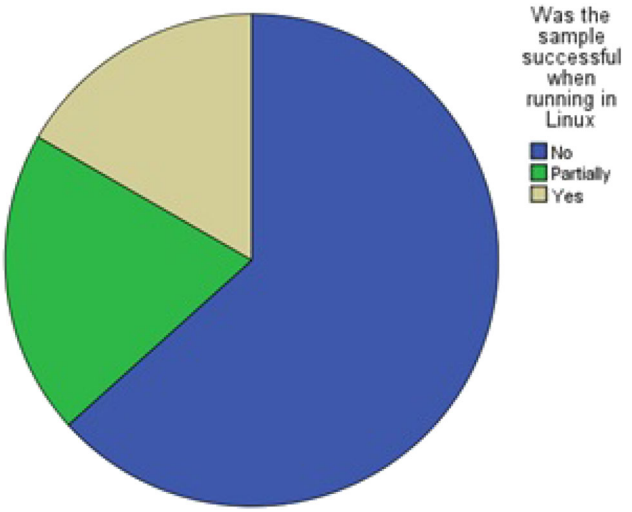


Fig. 3 The success rate of windows malware in wine

(n = 5) successfully ran on the Linux system, 20.0% had partial success (n = 6) and 63.3% (n = 19) failed to run in Wine. Table 61 shows the combined analysis results from all samples. Table 62 shows the combined details of the samples. This data set was used to test for trends between types of malware or malware behaviors and the malware being successful at running in Linux.

5.3 Hypotheses testing

To assess the hypothesis created, regression analysis was used in order to determine if there were any relationships between a malware sample's characteristics and its ability to run successfully in Linux. The regression analysis selected for this data sample was ordinal logistic regression. This method of regression analysis was the most suited to the data as it allows the use of one dependant variable and multiple independent variables. This allowed the success of the malware in Linux to be tested against certain malware characteristics and behaviours.

The dependent variable used for this test was the malwares success in Linux, defined as Yes, Partially or No. The ordinal regression statistical test could be used with this dependant variable as it allowed the use of this ordinal variable: where the relative ordering between categories is established as Yes>Partially>No. The aim of using this test was to determine if any specific types or behaviours of malware had a significant effect on the malwares ability to run in Wine. The independent variables selected to be used in this regression analysis were:

- The malware making changes to the registry in Windows
- The malware starting processes in Windows
- The malware starting services in Windows
- The malware initializing network traffic in Windows
- The number of files dropped by the malware in Windows
- The number of processes spawned by the malware in Windows
- The file size of the malware in bytes
- The compile date of the malware

Some independent variables that were intended to be included in this test were excluded due to the sample size that was collected. The independent variables that could not be included were:

- The malware making changes to the file system in Windows
- The category of malware

Ordinal regression allows the independent variables to be factors or covariates. The data collected holds a range of continuous variables such as the file size of a malware sample and the number of process spawned by the malware. These independent variables would be classed as covariates. There were also a range of categorical independent variables such as the malware making changes to the registry and the malware initializing network traffic. The categorical independent variables would be classed as factors in the regression analysis.

The regression analysis was used to test for strong relationships between any of these independent variables and the success of the malware (with large effect sizes, as hypothesised and per the low sample size). The model fitting results concluded that the p-value of this test was .912, the threshold for significance in this statistical analysis was a value of .05. The value of .912 exceeds this threshold, this value shows that the model does not give a significant improvement in terms of predicting the results based on the data that has already been collected. The p-value of .912 reveals that the results have a probability of 91.2% chance of being random. This suggests that the independent variables that were included in the ordinal regression test are very unlikely to have had any significant effect on the outcome of the results.

In the goodness of fit model, Pearson has a p-value of .236 and Deviance has a p-value of .439. The significance threshold value of these two p-values was again .05. Both of these p-values are significantly higher than the threshold in

Table 61 Overall malware success details

Malware family	File system	Registry	Processes	Network	Services success	Overall success
Narilam	True	True	True	N/A	N/A	True
Hikit	False	False	False	False	False	False
Stabuniq	N/A	N/A	False	False	False	False
Drixed	False	False	False	False	False	False
Batch Wiper	False	True	False	N/A	N/A	Partially
Dialer	False	False	True	N/A	N/A	False
MyDoom	False	False	True	False	False	False
Minamps	True	False	True	True	False	Partially
PlugX, Korplug	False	False	True	False	N/A	False
Wykcores	True	True	True	True	True	True
Didrex	False	False	False	False	False	False
Dozmot	False	N/A	False	N/A	N/A	False
Potao	False	N/A	True	N/A	N/A	False
Gamarue	False	True	False	False	N/A	Partially
TDL/Alureon	False	N/A	False	False	N/A	False
SC-KeyLog	True	True	True	True	N/A	True
Wirenet	True	N/A	True	True	N/A	True
CoreBot	False	False	False	False	N/A	False
Kawpfuni	True	False	True	N/A	N/A	Partially
Skypii	False	N/A	False	N/A	N/A	False
4DW4R3	True	N/A	True	False	N/A	Partially
LokiBot	False	False	False	N/A	N/A	False
Nitol	True	True	True	True	True	True
Nivdort	False	False	False	False	False	False
Unknown1	False	False	True	N/A	N/A	False
Unknown2	False	False	False	N/A	N/A	False
Unknown3	False	False	False	False	N/A	False
Unknown4	True	False	False	False	N/A	Partially
Unknown5	False	False	False	False	False	False

this instance. These results suggest that the model used does not have a good fit.

The Pseudo R-Square, Nagelkerke value is .103 this suggests that 10.3% of the outcome variants in the dependent variable are dependent on the independent variables used in this test. This suggests that the independent variables had very little impact on the outcome variants in the dependant variable. This value of .103 indicates that the particular malware behaviours and characteristics selected for this regression analysis had an insignificant effect on a malware samples' ability to run successfully in a Linux environment. Unfortunately this does not support the stated hypothesis.

None of the independent variables produced p-values below the significance threshold of .05, which suggests that from the results gathered none of the independent variables can be used to reliably predict the dependent variable. This indicates that there was not a statistically significant effect on the malware running successfully in a Linux environment by malware characteristic or behaviour.

6 Discussion

The purpose of this study was to assess the security implications of running Windows software through Wine on a Linux system. In order to gain insight into the practical magnitude of the issue, samples of malware were run in a Windows and Linux environment. The results gathered in both the Windows and Linux environments were compared.

The findings can be used to help provide an answer to what the security implications of using the compatibility layer software Wine are. It is clear that some malware samples were able to run completely successfully and others were able to run partially successful. This demonstrates that malware is

Table 62 Overall malware details

Family	Malware type	File name	Size	File type	Compile date
Narilam	Worm	data.exe	1639284	PE32 (GUI) Intel 80386	03/09/2009
Hikit	Rootkit	oci.dll	262656	PE32 (DLL) (console) Intel 80386	20/06/2011
Stabuniq	Trojan	F31B797831B3(...)	79360	PE32 (GUI) Intel 80386	21/03/2012
Drixed	Trojan	43s5d6f7g.exe	196608	PE32 (console) Intel 80386	14/12/2015
Batch Wiper	Trojan	GroveMonitor.exe_	185928	PE32 (GUI) Intel 80386	04/01/2011
Dialer	Spyware	91f805aece0434(...)	14720	PE32 (GUI) Intel 80386 (stripped)	08/06/2005
MyDoom	Worm	weaver.email.File(...)	34612	PE32 (GUI) Intel 80386	01/01/1970
Minamps	Trojan	Minaps_C99FA83(...)	647599	PE32 (GUI) Intel 80386	16/08/2009
PlugX, Korplug	RAT	Plugx_00fdb6a(...)	241622	PE32 (GUI) Intel 80386	15/02/2010
Wykcores	Trojan Dropper	Wykcores_0D38(...)	73484	PE32 (GUI) Intel 80386	19/06/1992
Didrex	Trojan	34frgegrg.exe	314880	PE32 (GUI) Intel 80386	03/02/2016
Dozmot	Trojan Dropper	Dozmot.D2190d(...)	30208	PE32 (DLL) (console) Intel 80386	04/05/2011
Potao	Backdoor Trojan	Potao_1st(...)	59904	PE32 (GUI) Intel 80386	01/10/2006
Gamarue	Worm	Gamarue.F.exe	32768	PE32 (GUI) Intel 80386	30/04/2005
TDL/Alureon	Rootkit	DNSChanger(...)	186368	PE32 (GUI) Intel 80386	09/03/2005
SC-KeyLog	Trojan Dropper	SCKeyLog.O_b(...)	29460	PE32 (GUI) Intel 80386	15/09/2004
Wirenet	Password Trojan	Host.exe	61952	PE32 (GUI) Intel 80386 (stripped)	07/08/2012
Dyzap	Spyware Trojan	Document-772976(...)	246784	PE32 (GUI) Intel 80386 (stripped)	22/05/1983
CoreBot	Banking Trojan	781c6.exe	497152	PE32 (GUI) Intel 80386	08/12/2015
Kawpfuni	Backdoor Trojan	1f0469a0(...)	585728	PE32 (GUI) Intel 80386	13/06/2009
Skypii	Worm	6B8E96CC(...)	103424	PE32 (GUI) Intel 80386	06/06/2004
4DW4R3	Rootkit	Rootkit_4rw3r3(...)	52224	PE32 (GUI) Intel 80386	27/01/2010
LokiBot	Spyware	4213294.root_1_0(...)	156672	PE32 (GUI) Intel 80386	07/06/2014
Nitol	Trojan	Yy999.exe	24576	PE32 (GUI) Intel 80386	08/09/2013
Nivdort	Trojan	sample.exe	892416	PE32 (GUI) Intel 80386	10/07/2013
Unknown1	Trojan Dropper	newversion.exe	5766144	PE32 (GUI) Intel 80386 (stripped)	01/11/1971
Unknown2	Unknown	647129ff9f491c(...)	301125	PE32 (GUI) Intel 80386 (stripped)	29/01/2012
Unknown3	Unknown	firefox.exe	1138688	PE32 (GUI) Intel 80386	29/01/2015
Unknown4	Unknown	ozze.exe	893942	PE32 (GUI) Intel 80386	29/10/2012
Unknown5	Unknown	host9.exe	76800	PE32 (GUI) Intel 80386	19/06/1992

able to run successfully in a Linux environment when run using Wine. This is clearly a security issue worth considering for Linux users who are using the software Wine to run arbitrary software. If malware is able to run via Wine, a Linux system can be compromised by malware that would otherwise be ineffective without Wine. The software Wine does not appear to have any warning messages during or after installation that could alert users to these security implications. A recommendation would be for Wine to include a message to users during installation that informs users about the importance of running verified software and using anti-malware software.

The table of results shows that the samples analyzed in this project indicated that generally malicious samples were fairly likely to be unsuccessful when running in Wine. As documented in the overall results in Table 61, two-thirds of the malware samples tested in this study were unsuccessful at running in Wine. This suggests that the majority of malware samples that are run in Wine will not run successfully. Table 61 also demonstrates that six of the samples tested were able to run partially successfully in the Linux environment. Thus demonstrating that some samples will run successfully in some areas but fail in others. An example of a partially successful sample can be viewed in Table 16, the sample Minamps was successful in terms of the changes made to the file system, the processes spawned and the network activity, however, it did not successfully make changes to the registry or start the required services and so was only partially successful.

Partially successful samples could potentially still pose threat to a Wine user, the malware's behavior may not be an exact copy of the behavior documented in the Windows envi-

ronment but the fact that some similar calls were found could still be a threat to the security and integrity of a computer system. For example, the system may not successfully start a service but could still fingerprint the system and send these details to a malicious server over the Internet. Table 16 shows that Minamps had successfully recreated the same network traffic as in Windows, which could have sent vital computer information to a remote server, resulting in Minamps being a serious threat to Wine users even though it was partially successful. Table 61 shows that eighteen samples were unsuccessful. The fact that a sample has failed also does not mean it is entirely harmless, malicious processes could stay running and cause damage to the computer.

The area with the lowest success rate in Linux was the ability for malware to start the same services as were started in the Windows environment. Referring to Table 61, it is clear that only two of ten malware samples were able to successfully recreate the services that were started in Windows in the Linux environment. This implies that malware that attempt to start services in the Linux environment may be more likely to fail or only be partially successful.

An ordinal regression statistical test was conducted, the results of this test were inconclusive as none of the independent variables were found to have a significant effect on the malware running successfully in Linux. Although some aspects of analysis were automated, the sample size was dictated by the manual nature of the comparison. The sample size could be increased in further investigations to test for relationships with smaller effect sizes.

The results of the analysis stage may suggest that the high-level malware behavior characteristics (use of services, registry, files etc.) that were tested for effect on malware success in Wine may have been too generic, and that the specific details in the use of services, libraries, and APIs were most likely the determinate factor for malware success in Wine. For instance, the OpenServiceA and OpenServiceW functions were never successfully called in the Linux environment in any of the analyses that took place. This suggests that malware that calls functions to certain open services such as the Tapisrv and Rasman services are more inclined to fail at running in a Linux environment. However, more analysis would need to be conducted in order to confirm this relationship. This finding may be a result of Wine not being able to recreate the OpenServiceA and OpenServiceW functions, another possibility may be that the services being called are not available through Wine. This again supports the suggestion that malware that calls services in Linux are more inclined to fail or only be partially successful.

Although the statistical test was statistically inconclusive the results can inform potential relationships. It is clear that malware does have the ability to run successfully in Linux, as five of the samples were found to run correctly in Linux, calling very similar calls to the API calls detected in the Windows environment. Although this is a relatively small selection of the thirty malware samples used in this investigation this provides evidence that suggests that malware is capable of compromising a Linux computer system. Six of the samples were able to run partially on the Linux system. This essentially means that they were able to make similar system calls and changes in some areas but failed to do so in others: for example, a partially successful malicious sample may successfully recreate all network traffic and network-based API calls in Linux but fail make the same registry changes that were made by the malware on a Windows system. An example of this would be the malware successfully making changes to the file system but failing to make key registry changes. Nineteen malicious files completely failed to compromise the Linux system matching none of the changes to the file system, registry, network or services. This suggests that many malicious Windows executables will fail to run in Wine. Overall the findings do suggest that using Wine is a security threat to Linux users. Although the chances of malware infecting a Linux computer running Wine appear to be relatively small, the risk is still present and users should proceed with caution when running unknown files in Wine. The security risk imposed by Wine is enhanced by the fact that Wine is not sandboxed from users' resources, meaning that any software running in Wine could potentially access all of a Linux user's files.

Another possible relationship that may exist but could not be statistically shown was that of malware starting network traffic in Windows and being successful. This was the independent variable that got the lowest p-value; a larger sample size might detect an effect on the malware success; however this relationship could not be shown from the sample collected in this study.

A similar study, yet much smaller in scope was conducted by Matt Moen in 2005 and concluded similar results to the results generated in this study [9]. The writeup is somewhat vague but concludes that one of the five samples (20%) tested appeared to run successfully in the Linux environment, which is a comparable percentage. The results of the study conducted in this report are likely to be more reliable as a higher number of malicious samples were analyzed when compared to Moen's study.

7 Limitations

The primary limitation of the project was the number of malware samples analyzed. A more automated analysis (perhaps at the expense of thoroughness) could provide more data for statistical analysis. This could uncover relationships that were not established from the data gathered in this research. A further limitation was that only limited interactive live analysis of the infected environments was conducted. Live

analysis could have included gaining more concrete evidence of a malware sample's success in the Linux environment by attempting to connect to any backdoors that the malware may have opened on the infected system. The analysis environment had limited network connectivity, in order to protect the other computer systems connected to the network.

8 Conclusion

The research conducted in this study produced a series of results that can be used to develop an understanding of the behavior of Windows malware running in Linux via Wine. Results indicate that Windows malware is able to run successfully in a Linux environment through Wine. The success rates of Windows malware running in a Linux environment does appear to be relatively low. The fact that some samples of malware did run successfully illustrates that using the compatibility layer software Wine in a Linux environment does present a security risk to Linux systems, which would otherwise be secure against Windows malware. No relationships could be established between any types of malware or behavior of malware and the malware running successfully in the Linux environment; relationships between the services started in Windows and Network started in Windows independent variables may be investigated via future research and an increased sample size.

The findings suggest that samples which use particular API calls are less likely to run successfully. The OpenServiceA and OpenServiceW functions were never called in Zero Wine suggesting that using these calls can cause compatibility issues with Wine. Another possible reason for this could be that the services being opened are not available through Wine.

References

1. Ahmadi, M., Sami, A., Rahimi, H., Yadegari, B.: Feature: malware detection by behavioural sequential patterns. Comput. Fraud Secur. **8**, 11–19 (2013)
2. Choudhary, S., Vidyarthi, M.D.: A simple method for detection of metamorphic malware using dynamic analysis and text mining. Procedia Comput. Sci. **54**, 265–270 (2015)
3. CodeWeavers. Technical White Paper: Running applications under crossover: an analysis of security risks (2008)
4. Davis, M.A., Bodmer, S.M., Lemasters, A.: Hacking Exposed Malware and Rootkits: Malware and Rootkits Secrets and Solutions. McGraw-Hill Education, New York (2009)
5. Elisan, C.C.: Malware, Rootkits & Botnets: A Beginner's Guide. McGraw-Hill Education, New York (2012)
6. Liangboonprakong, C., Sornil, O.: Classification of malware families based on N-grams sequential pattern features. In: 2013 IEEE 8th Conference on Industrial Electronics & Applications (ICIEA), p. 777 (2013)
7. Ligh, M., Adair, S., Hartstein, B., Richard, M.: Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code, pap/dvdr edn. Wiley, Indianapolis (2010)
8. Malin, C.H., Casey, E., Aquilina, J.M.: Malware Forensics Field Guide for Windows Systems: Digital Forensics Field Guides. Syngress, Waltham (2012)
9. Moen, M.: Running Windows viruses with Wine (2005)
10. Provataki, A., Katos, V.: Differential malware forensics. Dig. Investig. **10**, 311–322 (2013)
11. Raymond, E.S.: The Cathedral & the Bazaar, 1st edn. O'Reilly Media, Beijing (2001)
12. Secfence. Malware Analysis Delhi NCR, Memory Forensics Malware Testing India, Mumbai, Pune, Bangalore—Structured approach to malware reporting Static analysis, Dynamic analysis, Memory Forensics & Packet analysis. CERT CIRT Teams - Secfence Technologies
13. Seo, S.-H., Gupta, A., Mohamed Sallam, A., Bertino, E., Yim, K.: Detecting mobile malware threats to homeland security through static analysis. J. Netw. Comput. Appl. **38**, 43–53 (2014)
14. Sharif, M., Lanzi, A., Giffin, J., Lee, W.: Impeding Malware Analysis Using Conditional Code Obfuscation
15. Shijo, P., Salim, A.: Integrated static and dynamic analysis for malware detection. Procedia Comput. Sci. **46**, 804–811 (2015)
16. Touchette, F.: The evolution of malware. Netw. Secur. **2016**(1), 11–14 (2016)
17. Vasilescu, M., Gheorghe, L., Tapus, N.: Practical malware analysis based on sandboxing. In: 2014 RoEduNet Conference 13th Edition: Networking in Education & Research Joint Event RENAM 8th Conference, p. 1 (2014)
18. Wang, J., Shih, P.C., Carroll, J.M.: Revisiting Linuss law: benefits and challenges of open source software peer review. Int. J. Hum. Comput. Stud. **77**, 52–65 (2015)
19. WineHQ. WineHQ: What is Wine?
20. ZeroWine (2008). http://zerowine.sourceforge.net/. Accessed December 2008